

DAM
Desarrollo de Aplicaciones Multiplataforma
2º Curso

AD
Acceso a Datos

UD 8
Programación de componentes
de acceso a datos
(Parte 1)

IES BALMIS
Dpto Informática
Curso 2019-2020
Versión 1 (12/2019)

UD8 – Programación de componentes de acceso a datos

ÍNDICE

1. Introducción
2. Creación de componentes
3. Generar empaquetado de aplicaciones
 - 3.1 Empaquetado de una aplicación
 - 3.2 Ejecución de un empaquetado JAR
4. Generar componentes y reutilización
 - 4.1 Empaquetado de un componente
 - 4.2 Reutilización de un componente
5. Proyectos Maven y Dependencias
 - 5.1 Proyectos Maven con Java
 - 5.2 Proyectos Maven en NetBeans
6. Componentes visuales
 - 6.1 JavaFX y SceneBuilder
 - 6.2 Swing
 - 6.3 Ejecución de aplicación de escritorio de tipo Window

1. Introducción

Un **componente** software es una clase creada para ser reutilizada y que proporciona una funcionalidad a una aplicación o servicio determinado, pudiendo en algunos casos ser accesible mediante interfaz pública.

Los componentes se definen por su estado que se almacena en un conjunto de propiedades, las cuales pueden ser modificadas para adaptar el componente al programa en el que se inserte. También puede tener un comportamiento que se define por los eventos ante los que responde y los métodos que ejecuta ante dichos eventos.

En Java disponemos de dos plataformas de computación:

- **Java SE:** Plataforma de computación formada por una máquina virtual de Java más una biblioteca de clases estándares en la que se pueden ejecutar aplicaciones java.
- **Java EE:** Plataforma de computación con el mismo planteamiento que Java SE, pero con una biblioteca de clases más amplia donde muchas de ellas están pensadas para aplicaciones empresariales.

Un **JavaBean** es un componente de software creado para ser ejecutado en Java SE que cumple con las siguientes especificaciones:

- Debe tener un **constructor vacío**
- Debe implementar la interfaz **Serializable**
- Las **propiedades/atributos** deben ser **privados**.
- Debe tener métodos **getters o setters** o ambos que permitan acceder a sus propiedades

Un JavaBean en NetBeans se crea desde:

- "Java => Java Application" o
- "Java => Java Class Library"

Un **EJB (Enterprise JavaBean)** es un componente de software creado par ser ejecutado en Java EE.

Un EJB en NetBeans se crea desde:

- "Java EE => Enterprise Application" o
- "Java EE => EJB Module"

Una **aplicación Java de escritorio (Desktop App)** está almacenada en un dispositivo y se ejecuta accediendo a ella a través de gestor de archivos de su SO.

Las App Desktop pueden ser de dos tipos:

- **De Consola:** serán ejecutadas con la utilidad **java.exe**
- **De Ventanas (Window):** serán ejecutadas con la utilidad **javaw.exe**

Una **aplicación java de tipo web (Web App)** está almacenada en un servidor y se ejecuta accediendo a ella a través de de un navegador mediante pro protocolo HTTP o HTTPS.

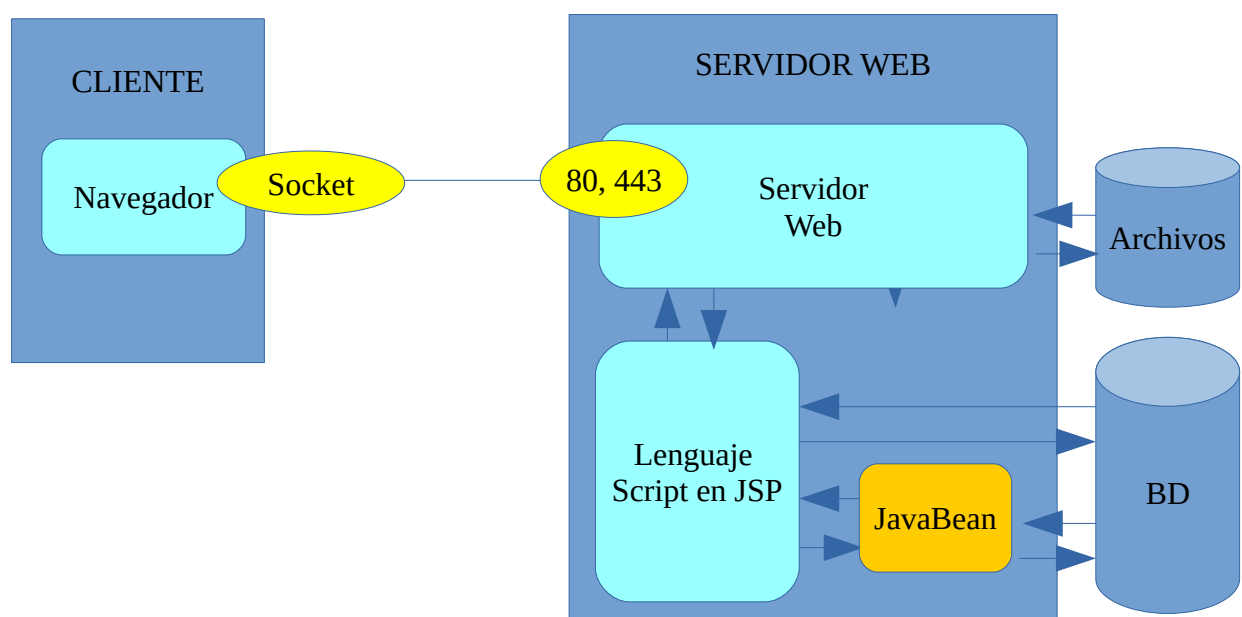
Las Web App de Java se desarrollan en utilizando el lenguaje script de servidor **Java Server Pages (JSP)**.

Los JSP pueden ejecutar código Java que genera páginas web dinámicas con tecnologías HTML+CSS+Javascript.

Un **Servlet** es un componente de Java utilizado para ampliar las capacidades de un servidor web, siendo utilizados por las páginas de tipo JSP que se ejecutan en el servidor.

Los **Servlets** son **JavaBean** pero para ser ejecutados en las páginas JSP y generalmente para implementar un **controlador**, es decir, un gestor de eventos y funcionalidad.

La mayoría de EJB y Servlets acceden a Bases de Datos, por lo que es necesario que tengan la funcionalidad de conexión al SGBD existente en el servidor.



2. Creación de componentes

Como en cualquier clase, un componente tendrá definido un estado a partir de un conjunto de atributos.

Los **atributos** son variables definidas por su nombre y su tipo de datos que toman valores concretos. Normalmente los atributos son privados y no se ven desde fuera de la clase que implementa el componente, se usan sólo a nivel de programación.

Las **propiedades** son un tipo específico de atributos que representan características de un componente que afectan a su apariencia o a su comportamiento. Son accesibles desde fuera de la clase y forman parte de su interfaz. Suelen estar asociadas a un atributo interno.

Una propiedad no es exactamente un atributo público, sino que tiene asociados ciertos métodos que son la única manera de poder modificarla o devolver su valor y que pueden ser de dos tipos:

GETTER

Permiten leer el valor de la propiedad y tienen la estructura:

```
public <TipoPropiedad> get<NombrePropiedad>( )
```

Si la propiedad es booleana el método getter se implementa así:

```
public boolean is<NombrePropiedad>()
```

SETTER

Permiten establecer el valor de la propiedad y tienen la estructura:

```
public void set<NombrePropiedad>(<TipoPropiedad> valor)
```

Si una propiedad no incluye el método set entonces es una propiedad de solo lectura.

Por ejemplo, si estamos generando un componente para almacenar datos de una **libro**, podemos tener, entre otras, una propiedad de sea **título**, que tendría asociados los siguientes métodos:

```
public void setTitulo(String titulo)
```

```
public String getTitulo()
```

Un **componente NO necesita** disponer de un método **main**, ya que se utilizará en otras aplicaciones.

Un **componente** bien definido debe incorporar al menos:

- La posibilidad de **Serializar** para conseguir la **persistencia** implementando la clase predefinida `java.io.Serializable` o `java.io.Externalizable`
- Las propiedades que almacenar
- Los constructores
 - Constructor sin parámetros
 - Constructor con todos los parámetros
 - En caso de existir una propiedad clave, el constructor de la dicha propiedad que no se repite
 - Los métodos getters y setters
 - El método `toString` sobrecargado para poder serializar a String
 - En caso de existir propiedades indexadas (`ArrayList`, `Set`, ...) es recomendable:
 - Añadir método para incorporar un elemento a la lista/conjunto
 - Añadir método para eliminar un elemento a la lista/conjunto

El **componente** debe incluirse en un **paquete** que posteriormente nos permitirá realizar el **import**.

Para ilustrar cómo se implementa un componente con una herramienta como NetBeans, crearemos uno proyecto de tipo "Java => Java Application" que permita almacenar la información de una **persona**, guardaremos por ejemplo, su nombre, apellidos, teléfono y dirección:

- De tipo `int` tendremos:
 - **dni**
- De tipo `String` tendremos:
 - **nombre, apellidos, telefono**
- La **domicilio** será una propiedad compuesta, por la dirección, código postal, la población y la provincia.

Dado que la última propiedad no es simple, será necesario crear otro componente **domicilio** que la desarrolle, y del que dependerá nuestro componente principal **persona**.

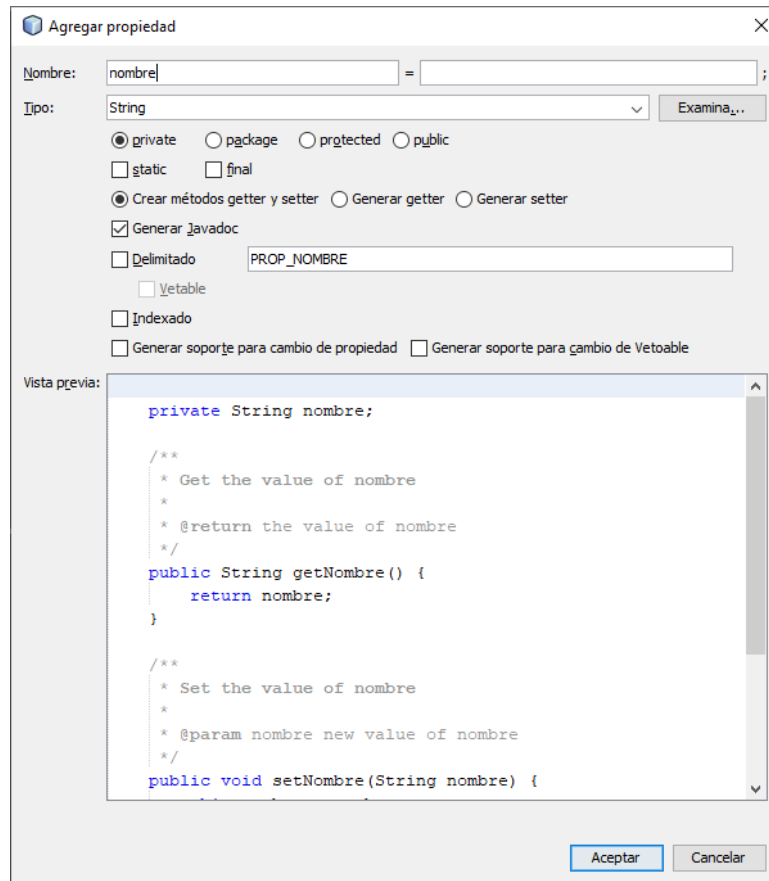
El proyecto contendrá la clase **public Persona**, y la clase **public Domicilio** .

Netbeans nos ofrece generación de código entre otros aspectos para:

- Constructores
- Getters y Setters
- `toString()`

Para ello, basta con pulsar en el editor con botón derecho y seleccionar "**Insert code**".

También podemos utilizar el asistente para añadir una propiedad con su setter y getter seleccionando "**Agregar propiedad**". Nos aparecerá en siguiente formulario:



Agregar propiedad

Nombre: = ;

Tipo:

☒ private ☐ package ☐ protected ☐ public
☐ static ☐ final
☒ Crear métodos getter y setter ☐ Generar getter ☐ Generar setter
☒ Generar Javadoc
☐ Delimitado
☐ Vetable
☐ Indexado
☐ Generar soporte para cambio de propiedad ☐ Generar soporte para cambio de Vetable

Vista previa:

```

private String nombre;

/**
 * Get the value of nombre
 *
 * @return the value of nombre
 */
public String getNombre() {
    return nombre;
}

/**
 * Set the value of nombre
 *
 * @param nombre new value of nombre
 */
public void setNombre(String nombre) {
  
```

Los componentes visuales incorporan además varios recursos como:

- Un editor de propiedades que implementa la clase predefinida **PropertyEditor**
- La reacción ante **eventos** para capturar y procesar las acciones que se producen

Una de las principales características de un componente es que una vez instalado en un entorno de desarrollo, éste debe ser capaz de identificar sus propiedades simplemente detectando parejas de operaciones get/set, mediante la capacidad denominada **introspección**.

3. Generar empaquetado de aplicaciones

El empaquetado de código compilado en Java implica la creación de un archivo binario.

El archivo de un código empaquetado de Java pueden tener diferentes extensiones como JAR, WAR o EAR.

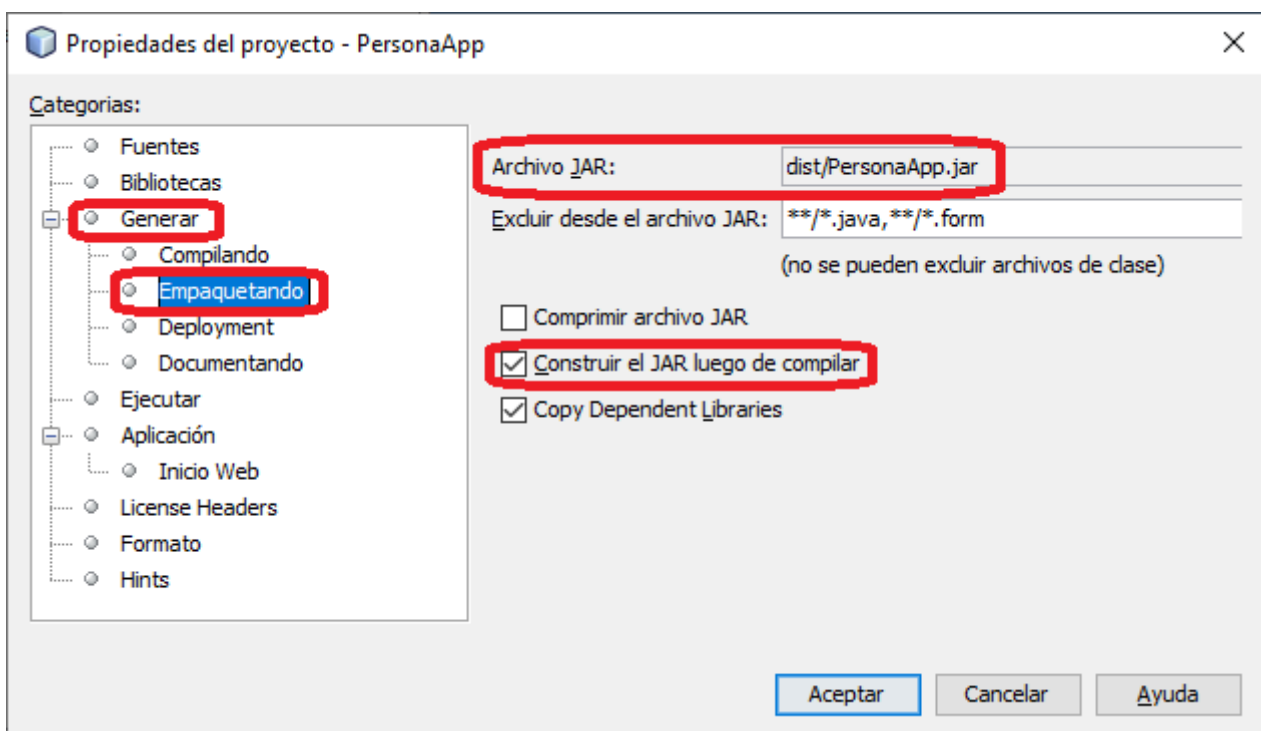
Nuestros primeros proyectos serán de extensión **JAR**.

Veamos un ejemplo:

Crear una aplicación en Java que muestre un texto por consola.
Utilizaremos para ello `System.out.println`

3.1 Empaquetado de una aplicación

Una vez realizado el proyecto, lo primero es activar el empaquetado automático al compilar en propiedades del proyecto:



En este ejemplo, el proyecto **PersonaApp** creará el empaquetado **PersonaApp.jar** en la carpeta **dist**.

Para empaquetar un componente en NetBeans debemos generar el proyecto con el icono martillo o la tecla F11:



3.2 Ejecución de un empaquetado JAR

Una vez hayamos obtenido el archivo JAR lo podremos ejecutar.

En nuestro caso, al ser una aplicación de escritorio de tipo consola, deberemos utilizar la utilidad **java.exe** con el siguiente comando:

```
java -jar PersonaApp.jar
```

Hay que tener en cuenta que si **java.exe** no se encuentra en el PATH de sistema, tendremos que indicar el camino completo. Por ejemplo:

```
C:\Program Files\Java\jre1.8.0_231\bin\java -jar PersonaApp.jar
```

4. Generar componentes y reutilización

4.1 Empaquetado de un componente

El empaquetado no está reservado exclusivamente a las aplicaciones, pues también se pueden crear componentes y empaquetarlos para poder reutilizarse en otros proyectos.

Veamos un ejemplo:

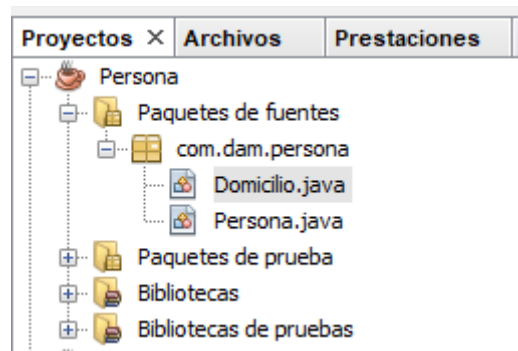
Crear un componente en Java que contenga:

- una clase `Domicilio` que almacene `direccion(String)`, `cpostal(String)`, `poblacion(String)` y `provincia(String)`
- una clase `Persona` que almacene un `dni(int)`, `nombre(String)`, `apellidos(String)`, `telefono(String)` y `domicilio(Domicilio)`

Las características del proyecto serán:

TIPO	Java Class Library (Es un Java Application sin método main)
PROYECTO	Persona
JAVA PACKAGE	com.dam.persona
CLASES	Serializables, Constructores, Getters y Setters, toString()

En NetBeans quedará:



Una vez activada la creación del JAR en propiedades al compilar, pulsamos en el martillo y generamos el JAR que se encontrará en la carpeta dist. El archivo tendrá el nombre de:

Persona.jar

4.2 Reutilización de un componente

Ahora crearemos una aplicación que use el componente creado.

Veamos un ejemplo:

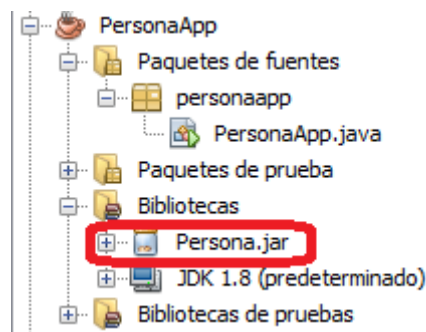
Crear una aplicación en Java que:

- Cree una instancia de **Persona** utilizando los constructores de **Domicilio** y **Persona**
- Mostrar su contenido utilizando **toString**

Crearemos un proyecto de tipo Java Application denominado **PersonaApp**.

Crearemos la carpeta **lib** y copiaremos el archivo **Persona.jar** del componente.

En NetBeans quedará:



El código de nuestro proyecto podría ser:

PersonaApp.java

```
package personaapp;

import com.dam.persona.Domicilio;
import com.dam.persona.Persona;

public class PersonaApp {

    public static void main(String[] args) {

        Persona persona = new Persona(21444555, "Sergio", "Fernández", "61277788",
            new Domicilio("C/Jaume de Scals, 35", "03100", "Xixona", "Alicante"));

        System.out.println(persona.toString());
    }
}
```

Podemos ver que el componente **Persona.jar** nos da la posibilidad de realizar import de sus clases **com.dam.persona.Domicilio** y **com.dam.persona.Persona**, como si las hubiéramos definido en el propio proyecto.

5. Proyectos Maven y dependencias

Existen multitud de componentes que podemos incorporar a nuestras aplicaciones y muchos de ellos dependen de otros, por lo que cuando un proyecto se hace grande y además hay actualizaciones es complejo gestionar el mantenimiento.

Por esta razón aparecen herramientas con repositorios de componentes. Para Java existen varios, pero los más usados son **Maven** y **Gradle**.

Gracias a estas herramientas, podemos añadir componentes y su documentación a nuestras aplicaciones de una manera sencilla, actualizada y segura.

5.1 Proyectos Maven con Java

Apache Maven Project es una herramienta de gestión y comprensión de proyectos de software basado en el concepto de un modelo de objeto de proyecto (POM).

Maven puede administrar la construcción, los informes y la documentación de un proyecto a partir de una información central.

Apache Maven Project

<https://maven.apache.org/>

Maven ofrece la posibilidad de explorar online los componentes pudiendo consultar diferente información organizada por **categorías** como: estadísticas de uso, revisiones y actualizaciones, tipos de licencia, fecha de publicación, ...

Repository Maven

<https://mvnrepository.com/>

<https://mvnrepository.com/repos/central> (para descargas directas de los JAR)

5.2 Proyectos Maven en NetBeans

MavenProject.java

Crea un proyecto con el contenido del proyecto **Mongo01Conexion** de la unidad didáctica de MongoDB pero con Maven

Para crear un proyecto con Maven utilizaremos "**Maven => Java Application**".

El asistente nos permitirá añadir información al POM del proyecto mediante un formulario. Es muy importante el ID del Grupo porque será el Java Package que identificará nuestro proyecto.

Nuevo Java Application

Pasos

1. Seleccionar proyecto
2. **Name and Location**

Nombre y ubicación

Nombre del Proyecto: MavenProject

Ubicación del Proyecto: C:\Users\USUARIO\Documents\NetBeansProjects [Examinar...]

Carpeta del proyecto: C:\Users\USUARIO\Documents\NetBeansProjects\MavenProject

Id del Artefacto: MavenProject

Id del Grupo: **com.dam**

Version: 1.0-SNAPSHOT

Paquete: com.dam.mavenproject (Opcional)

< Atrás Siguiente > **Terminar** Cancelar Ayuda

Crear clase principal

Si es un proyecto que se va a ejecutar, necesitará una clase con el método main.

En nuestro ejemplo crearemos una clase "**Java => Java Main Class**" que llamaremos **principal.java**

También se podría crear un "Java Class" y luego añadirle el método main.

Indicar en el proyecto la clase principal y la codificación UTF8

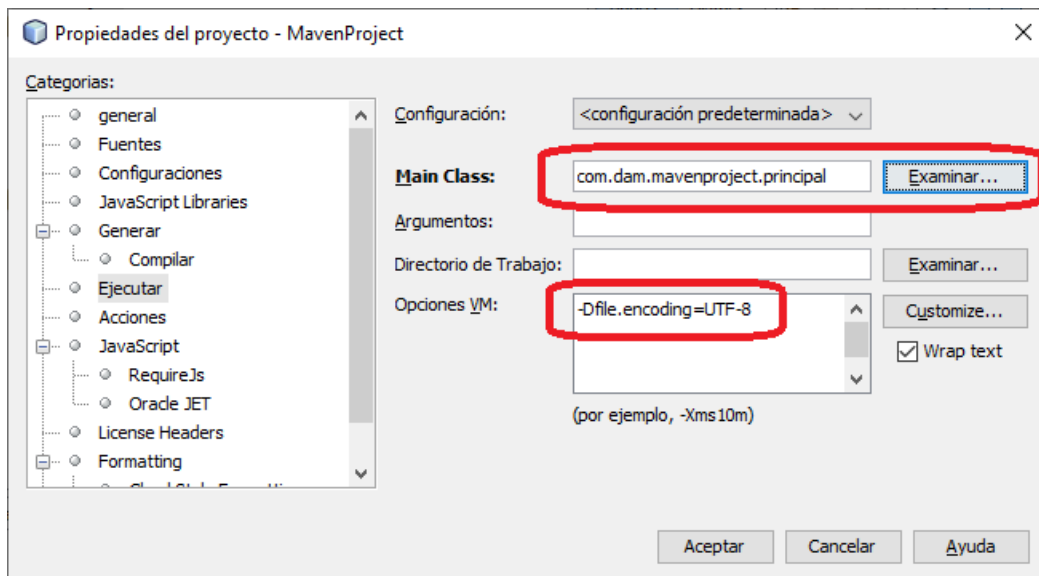
Pulsaremos con botón derecho sobre el proyecto => Propiedades

=> Ejecutar => Main Class

- Seleccionar la clase que contiene el main

- En Opciones VM añadir:

-Dfile.encoding=UTF-8



Añadir componentes y sus dependencias

Realizaremos un proyecto que conecta con MongoDB.

Las librerías o componentes están disponibles de forma online, por lo que es necesario disponer de conexión a internet.

ANEXO Maven Proxy

Maven necesita hacer descargas desde sus repositorios. En el caso de tener proxy, habrá que configurar Netbeans para que lo utilice.

El archivo a configurar es:

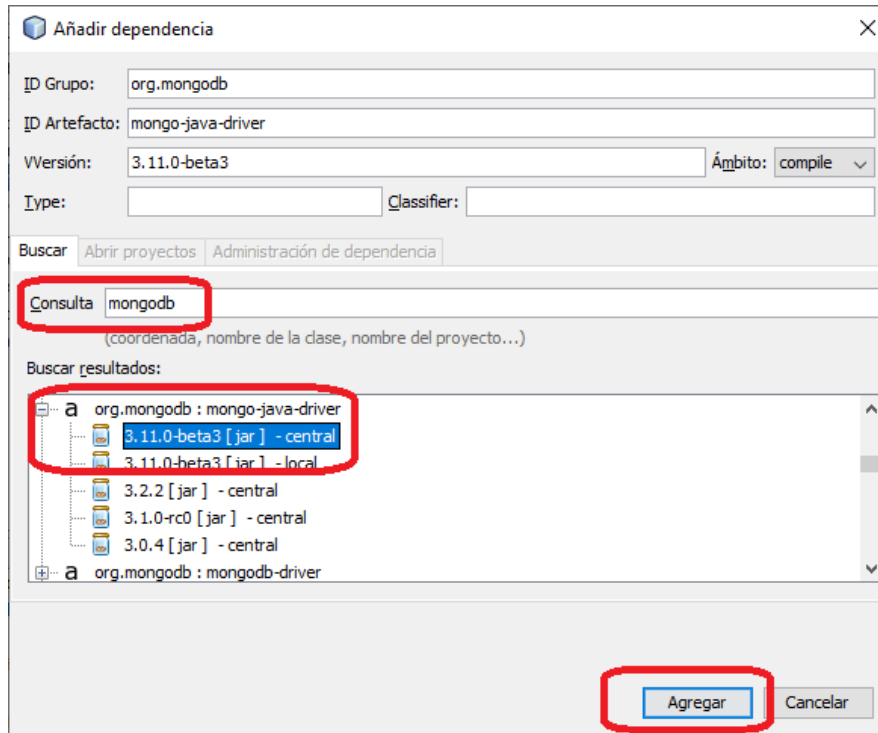
C:\Program Files\NetBeans 8.2\java\maven\conf\settings.xml

donde sustituiremos el bloque **proxies** por el siguiente:

```
<proxies>
  <!-- proxy
  | Specification for one proxy, to be used in connecting
  | to the network.-->
  <proxy>
    <id>optional</id>
    <active>true</active>
    <protocol>http</protocol>
    <username>alumno</username>
    <password>alumno</password>
    <host>192.168.0.100</host>
    <port>8080</port>
    <!-- <nonProxyHosts>local.net|some.host.com</nonProxyHosts> -->
  </proxy>
  <!-- -->
</proxies>
```

Pulsando con botón derecho en "Dependencias => Agregar dependencias"

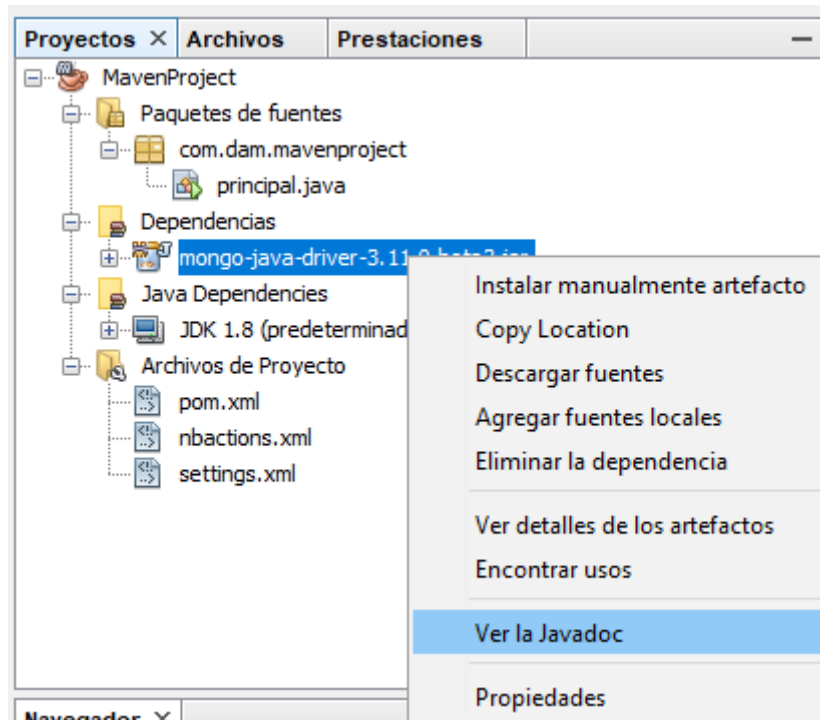
- En "Consulta" escribir **mongodb** y buscar **org.mongodb:mongo-java-driver**
- Pulsar en [+]
- Seleccionar 3.11.0-beta3[jar]
- Agregar



Al añadir un componente, Maven incorpora todas sus dependencias, por lo que no tenemos que preocuparnos por el resto de archivos (recuerda que cuando lo añadimos manualmente utilizábamos 6 archivos JAR)

Consultar la documentación o manual de referencia

Con botón derecho sobre el componente jar => Ver la JavaDoc, se accederá localmente al manual de referencia.



Maven abrirá un servidor web virtual local en el puerto 8082, y navegaremos sobre la documentación oficial.

Información sobre los componentes

Si queremos consultar la información disponible de cada componente añadido, pulsaremos con botón derecho sobre el jar => Ver detalles de los artefactos, podremos tener acceso a datos sobre:

- Básicos
- Proyectos
- Ruta de clases

Código de nuestra aplicación

Solo falta incluir en nuestra aplicación el código y probar a ejecutarlo. Añadiremos como prueba el código siguiente:

principal.java

```
package com.dam.mavenproject;

import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.bson.Document;

public class principal {
    public static void main(String[] args) {

        // Desactivar logs de MongoDB
        Logger mongoLogger = Logger.getLogger("org.mongodb.driver");
        mongoLogger.setLevel(Level.SEVERE);

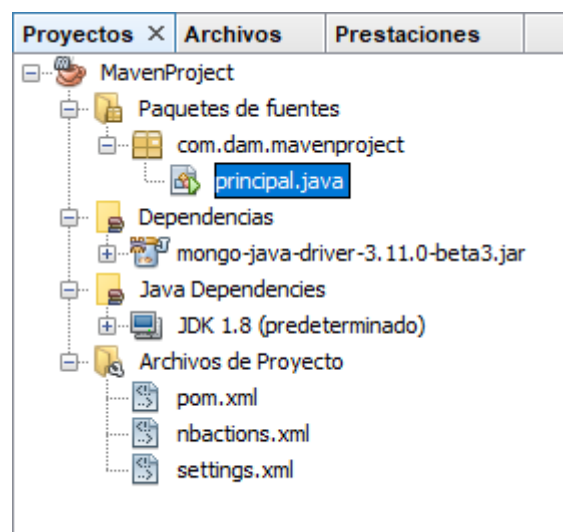
        System.out.println("Conectando a MongoDB.");

        // Conexión a MongoDB.
        MongoClient mongoClient = new MongoClient("localhost",27017);
        System.out.println("Seleccionando BD y colección.");
        MongoDatabase database = mongoClient.getDatabase("demografia");
        MongoCollection<Document> collection = database.getCollection("comunidades");

        // Información
        System.out.println("En la BD .....\'" + database.getName() + "\'");
        System.out.println("En la Colección ..\'" + collection.getNamespace() + "\' ");
        System.out.println("El número de documentos es " + collection.countDocuments() );

        // Desconexión
        System.out.println("Desconectando de MongoDB.");
        mongoClient.close();
    }
}
```

El proyecto quedará:



6. Componentes visuales

Como ya hemos comentado anteriormente, en Java se pueden crear aplicaciones de escritorio de tipo Ventana. Esto permite su ejecución en entornos gráficos abriendo ventanas en el escritorio del dispositivo.

Para crear estas ventanas Java dispone de varios componentes visuales. Los 3 más usados son:

- AWT
- JavaFX con SceneBuilder
- Swing

AWT

- Es la base de Swing, funciona bien pero carece de componentes avanzados.
- Si se tiene la intención de crear aplicaciones ricas, AWT probablemente no sea el camino a seguir.
- Podría usarse para aplicaciones GUI pequeñas que no requieran interfaces de usuario enriquecidas, ya que es un entorno muy probado.

SWING

- Está basado en AWT como se indicó anteriormente.
- Al principio, se consideraba lento y con errores, e hizo que IBM creara SWT para Eclipse. Sin embargo, con Java 6 Swing se convirtió en el GUI de elección para crear nuevas aplicaciones.
- Swing tiene muchos componentes enriquecidos, pero todavía faltan en algunas áreas. Un ejemplo es que no hay un componente TreeTable con todas las funciones que pueda ordenar y filtrar / buscar.

JavaFX

JavaFX es el GUI diseñado por Java/Oracle que promete ser el estándar de facto en el desarrollo de aplicaciones web o de escritorio enriquecidas.

- En JavaFX, los administradores de diseño son nodos
- JavaFX ha mejorado el manejo de eventos
- JavaFX admite propiedades
- JavaFX es parametriza el diseño con estilos de CSS
- JavaFX tiene controles más consistentes
- JavaFX tiene efectos especiales
- Las animaciones son más fáciles en JavaFX
- JavaFX es compatible con dispositivos táctiles modernos
- JavaFX no tiene equivalente a JOptionPane de Swing

Fuente: <https://www.dummies.com/programming/java/10-differences-between-javafx-and-swing/>

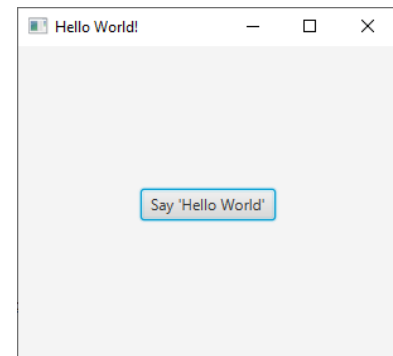
Todos estos componentes son similares, a la hora del diseño, pero cambian bastante en su uso por su diferente modelo de clases.

6.1 JavaFX y SceneBuilder

Si deseamos realizar aplicaciones profesionales portables acabaremos usando JavaFX.

En NetBeans podemos crear un proyecto **JavaFXApp1** de tipo "**JavaFX => JavaFX Application**".

Si lo probamos y ejecutamos nos aparecerá la ventana:



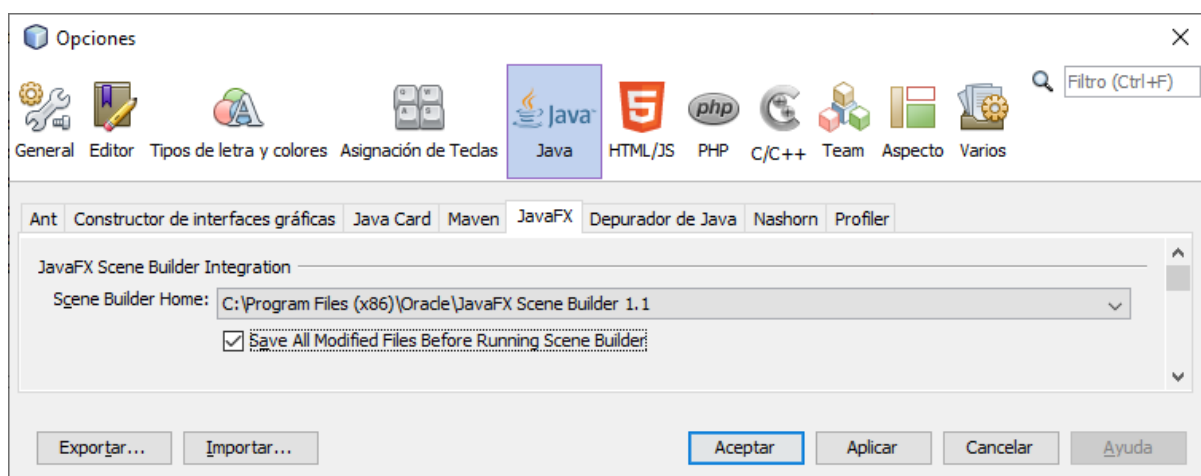
Si queremos disponer de un diseñador gráfico tenemos que instalar la herramienta SceneBuilder.

SceneBuilder Web Oficial

<https://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html>

Una vez descargado e instalado, deberemos indicar a NetBeans en qué carpeta se encuentra.

Accederemos a "Herramientas => Opciones => Java => JavaFX" y seleccionar la carpeta de instalación de SceneBuilder:



Crearemos un proyecto **JavaFXApp2SceneBuilder** donde podamos utilizar SceneBuilder en el diseño de las ventanas seleccionando:

"JavaFX => JavaFX FXML Application".

Haciendo doble-click sobre el archivo con extensión fxml y se abrirá la herramienta ScenBuilder para editarlo.

Si deseamos una ventana nueva, utilizaremos **"JavaFX => Empty"**, marcando que nos añada el Controlador. El controlador de cada ventana tendrá la clase con los métodos que atenderán los eventos producidos.

JavaFX más información

https://docs.oracle.com/javafx/scenbuilder/1/use_java_ides/sb-with-nb.htm

6.2 Swing

Para crear un proyecto con Swing seleccionaremos **"Java => JavaApplication"**.

Swing tiene dos formas de trabajo:

- Creación dinámica (en tiempo de ejecución) de elementos visuales
- Creación visual de ventanas

Veamos ejemplos.

SwingDinamico

Aplicación que pida el nombre de un archivo, por ejemplo "prueba" y cree un archivo llamado "prueba.txt" en la carpeta "./datos/" con el texto:

"Este archivo se crea desde una aplicación de escritorio de tipo Window"

A continuación si no hay error mostrará un mensaje **"Archivo creado"** y en caso contrario **"Error al crear el archivo"**.

```
public class SwingDinamico {
    public static void main(String[] args) throws FileNotFoundException {

        String archivo = JOptionPane.showInputDialog("Introduce el nombre de un archivo: ");
        try {
            PrintWriter fichero = new PrintWriter("./datos/"+archivo+".txt");
            fichero.println("Creado desde la aplicación de escritorio de tipo Window");
            fichero.close();

            JOptionPane.showMessageDialog(null, "Archivo creado");

        } catch (FileNotFoundException e) {
            JOptionPane.showMessageDialog(null, "Error al crear el archivo");
        }
    }
}
```

JOptionPane – Ventanas de Diálogo (Message + Input)

<https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html>

La clase static **JOptionPane** permite mostrar ventanas de entrada de datos (**showInputDialog**) y ventanas de mensajes (**showMessageDialog**).

SwingVisual

Aplicación que pida seleccionar un archivo de texto y muestre su contenido en un control de tipo "area".

Cuando queramos añadir una ventana, utilizaremos archivo nuevo y el tipo deseado:

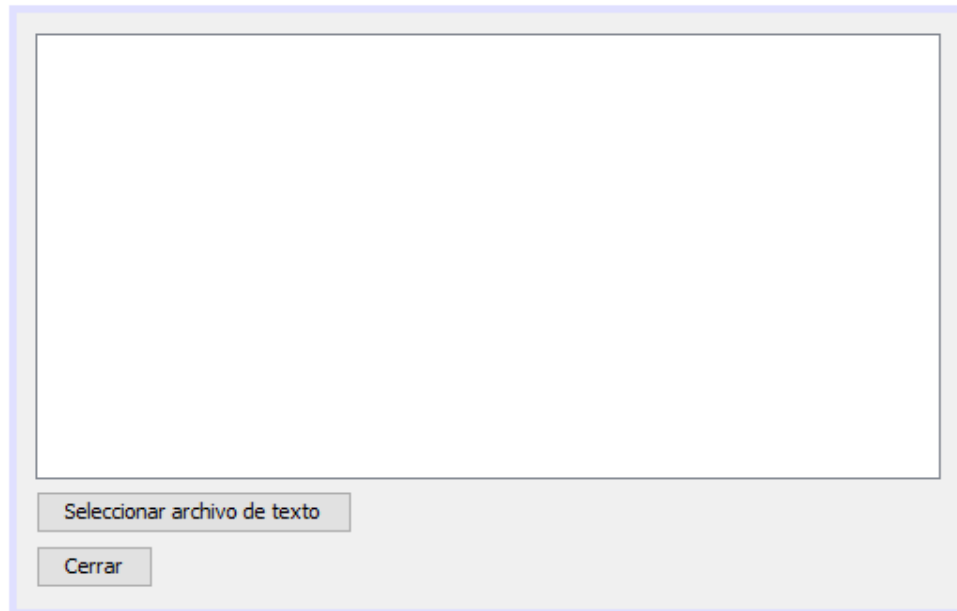
Formularios de interfaz gráfica de Swing =>

=> Formulario JDialog

=> Formulario JFrame

=> Formulario JPanel

En nuestro ejemplo crearemos un "**Formulario JFrame**" denominado **WinJFrame** y le añadiremos un `TextArea` (área de texto) y dos `Button` (Botón).



Ahora tenemos que introducir código en nuestra clase principal para llamar a la ventana:

```
public class SwingVisual {  
    public static void main(String[] args) {  
        java.awt.EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                JFrame ventana = new WinJFrame();  
                ventana.setTitle("Cargar visor de archivo de texto:");  
                ventana.setVisible(true);  
            }  
        });  
    }  
}
```

En el formulario añadiremos código al botón "Cerrar" haciendo doble click sobre él:

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}
```

Y al botón "Seleccionar archivo de texto":

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser jfc = new JFileChooser(new File("./datos/"));
    jfc.setDialogTitle("Elegir un fichero de texto: ");
    jfc.setFileSelectionMode(JFileChooser.FILES_ONLY);

    FileNameExtensionFilter filter =
        new FileNameExtensionFilter("TEXT FILES", "txt", "text");
    jfc.setFileFilter(filter);

    int returnValue = jfc.showOpenDialog(null);
    if (returnValue == JFileChooser.APPROVE_OPTION) {
        File selectedFile = jfc.getSelectedFile();

        try (BufferedReader ficheroEntrada =
            new BufferedReader(new InputStreamReader(
                new FileInputStream(selectedFile.getAbsolutePath()),
                StandardCharsets.UTF_8))) {

            jTextArea1.setText(null);
            String linea = null;
            while ((linea = ficheroEntrada.readLine()) != null) {
                jTextArea1.append(linea+"\n");
            }
            ficheroEntrada.close();
        } catch (IOException ex) {
            Logger.getLogger(WinJFrame.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

6.3 Ejecución de aplicación de escritorio de tipo Window

Una vez hayamos obtenido el archivo JAR lo podremos ejecutar haciendo doble click sobre él.

En nuestro caso, al ser una aplicación de escritorio de tipo Windows, se usa por defecto la utilidad **javaw.exe**. También podemos ejecutarlo con el siguiente comando:

```
javaw -jar SwingVisual.jar
```

Hay que tener en cuenta que si javaw.exe no se encuentra en el PATH de sistema, tendremos que indicar el camino completo. Por ejemplo:

```
C:\Program Files\Java\jre1.8.0_231\bin\javaw -jar SwingVisual.jar
```